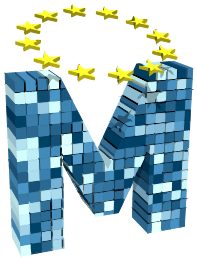


Project title: Multi-Owner data Sharing for Analytics and Integration respecting Confidentiality and OWNer control
Project acronym: MOSAICrOWN
Funding scheme: H2020-ICT-2018-2
Topic: ICT-13-2018-2019
Project duration: January 2019 – December 2021

D3.3
First Version of Policy Specification Language and Model

Editors: Sabrina De Capitani di Vimercati (UNIMI)
Pierangela Samarati (UNIMI)
Reviewers: Jonas Böhler (SAP SE)
Rigo Wenning (W3C)

Abstract

This deliverable presents the first version of the policy model and language developed in Work Package 3 (WP3). The main goal of WP3 is the definition of a data governance framework for managing data and for specifying policies in the data market scenario. Starting from all the relevant requirements and protection needs identified from the use cases as well as from the data market context, we have first identified different types of policies that can be applied during different stages of the use of the data market platform (i.e., ingestion, access, and sharing). The deliverable then focuses on the access policies and describes the main concepts of the model and of the language along with a first version of the policy engine enforcing the access policies.

Type	Identifier	Dissemination	Date
Deliverable	D3.3	Public	2020.06.30

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825333.

MOSAICrOWN Consortium

1.	Università degli Studi di Milano	UNIMI	Italy
2.	EMC Information Systems International	EISI	Ireland
3.	Mastercard Europe	MC	Belgium
4.	SAP SE	SAP SE	Germany
5.	Università degli Studi di Bergamo	UNIBG	Italy
6.	GEIE ERCIM (Host of the W3C)	W3C	France

Versions

Version	Date	Description
0.1	2020.06.04	Initial Release
0.2	2020.06.25	Second Release
1.0	2020.06.30	Final Release

List of Contributors

This document contains contributions from different MOSAICrOWN partners. Contributors for the chapters of this deliverable are presented in the following table.

Chapter	Author(s)
Executive Summary	Sabrina De Capitani di Vimercati (UNIMI), Pierangela Samarati (UNIMI)
Chapter 1: Introduction	Sabrina De Capitani di Vimercati (UNIMI), Pierangela Samarati (UNIMI)
Chapter 2: General principles and desiderata	Sabrina De Capitani di Vimercati (UNIMI), Sara Foresti (UNIMI), Stefano Paraboschi (UNIBG), Pierangela Samarati (UNIMI)
Chapter 3: Policy model	Sabrina De Capitani di Vimercati (UNIMI), Sara Foresti (UNIMI), Stefano Paraboschi (UNIBG), Pierangela Samarati (UNIMI)
Chapter 4: Policy language	Sabrina De Capitani di Vimercati (UNIMI), Sara Foresti (UNIMI), Stefano Paraboschi (UNIBG), Pierangela Samarati (UNIMI)
Chapter 5: Policy engine	Enrico Bacis (UNIBG), Sabrina De Capitani di Vimercati (UNIMI), Dario Facchinetti (UNIBG), Sara Foresti (UNIMI), Stefano Paraboschi (UNIBG), Matthew Rossi (UNIBG), Pierangela Samarati (UNIMI)
Chapter 6: Conclusions	Sabrina De Capitani di Vimercati (UNIMI), Pierangela Samarati (UNIMI)

Contents

List of Figures

List of Tables

Executive Summary

This deliverable reports on the first release of the MOSAICrOWN policy specification language and model. It illustrates different aspects to be considered in the specifications, including the different protection requirements coming from the use cases of the project, and illustrates the main concepts and features of our proposal. The MOSAICrOWN language and model enable data owners to effectively control their data ingested, stored, and processed in the data market.

Starting with a brief introduction (Chapter 1) summarizing the state-of-the-art and outlining the main innovation and advantages provided by our solution, the core of the document is then organized in four main chapters.

Chapter 2 identifies the basic principles and desiderata that should be considered in the design of the policy model and language. In particular, it analyzes the requirements identified by the three use cases of MOSAICrOWN, which introduce a comprehensive set of needs to be considered for effectively protecting data in the digital data markets. In the chapter, we analyze such requirements identifying their impact on the language and model with respect to data organization on one side, and policy specifications on the other side. Building on such analysis, we then identify the main basic principles and desiderata that the policy model and language should satisfy.

Chapter 3 defines the assumptions and the basic concepts for the policy model. In particular, it identifies the basic elements of the model, including subjects, datasets, metadata, operations, and purposes. It also discusses the inclusion of data protection aspects introducing the concept of data transformations, which comprise wrapping and sanitization. The support of data transformations in the model nicely provides control at the policy specification level of data protection techniques (including the wrapping and sanitization solutions designed in WP4 and WP5, respectively), enabling data owners to conveniently and effectively command their application as needed.

Chapter 4 illustrates the preliminary version of the MOSAICrOWN policy specification language. In the chapter, we first identify the different phases in the data life-cycles where policy may need to be considered, and illustrate then the basic constructs of the specification for regulating access to data in the digital data market. The language builds upon the policy model defined in the previous chapter supporting specification of policy rules in a simple, yet flexible and expressive way. In particular, the language provides support for conditions on subject profiles and metadata, and explicit consideration of purpose of access. Leveraging the abstractions supported by the policy model, the language supports the specification of policy rules at different granularity levels. The chapter also discusses other aspects related to policy specification and enforcement, which we have investigated in the project.

Chapter 5 illustrates the preliminary version of the policy engine, providing for effective enforcement of the policy specifications. In line with the project ambition and requirement to seek effective deployment and compatibility with current technology, our realization of the policy engine builds on ODRL (a W3C policy specification standard). We then illustrate how MOSAICrOWN policies can be translated into ODRL specifications, addressing then different problems that raise in the actual evaluation and enforcement of policy rules.

1. Introduction

The goal of MOSAICrOWN is to enable data sharing and collaborative analytics in multi-owner scenarios, providing owners with control on the information shared and released to others, together with efficient and scalable techniques for enforcing data protection and supporting privacy-preserving analytics. To this end, MOSAICrOWN addresses protection requirements of data in the different phases of the information life-cycle with the aim of providing a comprehensive data management framework realizing the envisioned goal and ambition. Work Package 4 (WP4) and Work Package 5 (WP5) develop core technologies for enforcing data protection, providing for solutions wrapping the data with a (reversible) self-protecting layer (WP4) and sanitizing the data to produce a privacy-preserving view on them (WP5). Work Package 3 (WP3) is devoted instead to the development of a data governance framework for MOSAICrOWN. Such a governance framework represents the actual means provided to data owners to maintain control on their data, including regulating the controlled application, and reasoning upon the data protection techniques designed in WP4 and WP5. At the core of the governance framework is the policy model and language, presented in this deliverable, dictating the policy specifications regulating data access, usage, processing, and release. The policy model and language provide then the glue, nicely bringing together and leveraging the availability of technical solutions in the project. This deliverable presents the preliminary version of the policy model and language, illustrating the different requirements guiding its design (including, in particular, requirements coming from the use cases) and presenting its basic concepts, specifications, as well as direction for implementation.

1.1 State-of-the-art

Data sharing and dissemination are basic features for data markets that should be supported in a controlled way so that data owners remain in control of their data. This is also in line with recent laws and regulations (e.g., the EU GDPR) that empower the subject of a data item (i.e., the individual to whom the data item refers) with rights over it. The consideration of these problems in the data market scenario introduces the need for supporting expressive and flexible policies, which impose restrictions on the use and processing of data, and efficient and effective enforcement mechanisms [DFLS20]. Such a problem is largely recognized and the growing interest in the data market platform clearly strengthens such need, calling for solutions that can regulate the use, processing, and dissemination of (potentially sensitive) data, enhancing the control of data owners on their data. Recognizing the importance and interest of these techniques, the research and industrial communities have investigated solutions for empowering users with control over their data in different sharing scenarios. Work related to the definition of policy language and model touches upon these works as well as works addressing generic access control (authorizations) solutions [DFS20]. Among the different aspects investigated, there are approaches aiming at: supporting privacy of users in digital interactions (e.g., [ACK+10, ADF+12, BS02]),

leveraging encryption for enforcing access control (e.g., [BDF+18, DFJ+10]), accounting for non fully trusted storage providers (e.g., [ZDX+20]), and enriching authorization specifications (e.g., [BK19, BS03, JSSS01]). Among related works, also work carried out in the context of European projects, such as PrimeLife (primelife.ercim.eu) and SPECIAL (www.specialprivacy.eu), specifically targeting policies and privacy. Such work considered however a different focus. In particular, Primelife was concerned with privacy of users accessing the system, in contrast to privacy of data accessed and processed as MOSAICrOWN. SPECIAL instead focused on the support of GDPR specifications, hence considering a limited set of requirements that may need to be considered in real-life digital data markets, which MOSAICrOWN addresses. Therefore, such works are complementary in focus and goal with respect to MOSAICrOWN. One of the main challenges in the design of a policy model and language is to balance simplicity and easy of use (to make it appealing and acceptable for end users as well as to ensure its effective and efficient enforcement) with its expressiveness and flexibility (to ensure its suitability for capturing different requirements). For instance, logic-based languages, while appreciable for their elegance and expressiveness, may result not suitable in practical settings for their complexity in use and enforcement. In MOSAICrOWN we address such a challenge, providing for a simple and easy to use, yet flexible and expressive, language supporting abstractions, conditional expressions on data and authorization subject, and explicit consideration of purpose of access and of data transformation (which trigger the application of data wrapping and sanitization).

1.2 MOSAICrOWN innovation

The work of MOSAICrOWN is proceeding towards the development of a *simple and easy to use, yet expressive, flexible and extensive policy language* that introduces and enjoys several advantages, including support for:

- different requirements that data owners might wish to specify, and have enforced, on data ingested, stored, or processed in the data market;
- different privacy concepts and conditions, including purpose, privacy metrics, and privacy degrees;
- specification of data transformations, including wrapping and sanitization, to provide protection of data with direct control in the language;
- policy rules at different granularity levels, with consideration of abstractions;
- data access and usage conditions depending on metadata associated with data and subjects;
- enforcement engine via translation of high-level (user friendly) specifications to standard solutions for providing effective deployment and interoperability with existing solutions.

2. General principles and desiderata

The goal of this chapter is to analyze the requirements of the use cases described in deliverable D2.1 "Requirements from the Use Cases" [WB19] and to identify concepts and needs that should be supported by the policy model and language. Our analysis then focuses on the requirements that may have an impact on the definition of the policy model and language. The chapter is organized in two sections: Section 2.1 presents, for each use case, the relevant requirements together with a comment on them; Section 2.2 describes a synthesis of the requirements listed in the previous section by highlighting the principles and desiderata that our model and language should satisfy.

2.1 Use cases

MOSAICrOWN has three use cases that represent real-world problems where the industrial partners are involved and for which there are open privacy/confidentiality problems. For each use case, we now report the list of requirements that may have an impact on the definition of our policy model and language together with a comment on them. In reporting the requirements, we distinguish them in two categories: *data* requirements impacting how datasets should be managed, stored, and processed; and *policy* requirements impacting the definition of policies. The reason for such categorization is to help our analysis on the requirements, addressing first the problem of defining and organizing data on which policies are specified, and then the definition of the policy model and language.

2.1.1 Protection of sensitive data in an ICV data market (UC1)

Use Case 1 falls in the automotive domain and aims at addressing the data privacy issues related to autonomous vehicles. The goal is the design of mechanisms (wrapping and sanitization) to protect sensitive portions of the collected data while retaining the utility of the data for use in a collaborative data market shared among multiple Original Equipment Manufacturers (OEMs). The use case mainly focuses on the ingestion phase, where the specification of policies and the application of data wrapping/sanitization techniques is fundamental for properly regulating access to sensitive information, and also touches subsequent phases (i.e., storage and analytics) of the data life-cycle.

Data

- REQ-UC1-DI4: Ingestion mechanism should support different data types and formats, and account for structured and unstructured data.
- REQ-UC1-DI5: Ingestion mechanism should implement data wrapping and sanitization functions according to the defined data governance model.

- REQ-UC1-DI7: Identifiers (e.g., VIN) should be preserved but secured from unauthorized access.
- REQ-UC1-DP1: Output from the analytics process can be stored on the platform, according to the data governance specification.
- REQ-UC1-DP2: Output from the analytics process should be sanitized according to the data governance specification.
- REQ-UC1-DP3: Output from the analytics process can feedback into the shared data set, enriching the data.

Policies

- REQ-UC1-DG1: A clear language and set of definitions for describing the data governance model are required to support the definition and enforcement of access control rules and data protection mechanisms on the data.
- REQ-UC1-DG2: The platform should provide the capability for data providers to define data governance models for new data sets entering the platform.
- REQ-UC1-DG3: Data protection parameters for wrapping and sanitization should be configurable by the data owner.
- REQ-UC1-AC1: Access control and authorizations mechanisms should support varying levels of granularity.
- REQ-UC1-AC2: It should be possible to grant and revoke access to specific data sets or fields.
- REQ-UC1-AC4: The platform should allow restricting some data sets and/or platform users (providers or consumers) to use the platform for sharing only, analytics only, or both analytics and sharing.
- REQ-UC1-AC5: The platform should allow data providers to share data with a particular data consumer.
- REQ-UC1-AC6: The platform should allow data providers to share data with multiple data consumers.
- REQ-UC1-AC7: Policies for data sets and platform users should be configurable by the data provider.

Comment. Use Case 1 provides structured and unstructured datasets and requires the application of wrapping/sanitization techniques at ingestion. Data owners can select the configuration parameters for sanitization and wrapping and ingest this information together with the datasets. After ingestion, datasets can be shared with a specific data market consumer (i.e., customers of the data market) or set of consumers. Access control rules can be specified at different granularity levels. Datasets can be distinguished between datasets for sharing and for analytics. The result of an analysis can be possibly sanitized and stored in the data market.

2.1.2 Data markets for analysis of financial data (UC2)

Use Case 2 considers sensitive financial data as well as customer behavior and demographic data whose security is of paramount importance. The main goal of this use case is to analyze financial transactions to produce market-level analytics and forecasts at both microeconomic and macroeconomic levels. The analytics functions can be performed both inside the data market and outside the data market.

Data

- REQ-UC2-DI2: Ingestion mechanism should support different data types and formats (e.g., alphanumeric, integer, floating).
- REQ-UC2-W6: Unique Identifiers (e.g., Customer Ids, Account numbers) should be anonymized at the source as well as re-anonymized in the repository.

Policies

- REQ-UC2-AC1: Data set, Table row, Table column level access controls will be implemented with the goal of giving customized privileges to data. This would be to control who could see individual data sets, attributes, and measures.
- REQ-UC2-AC2: User level access control management system will be built into the system. This will allow giving permissions to individual users based upon their roles and what they should or should not see. It also gives the flexibility to the system to add and remove users quickly and efficiently.
- REQ-UC2-AC3: Business/Organization level access control management system will be built into the system. This will allow giving base permissions to an organization as a whole. It also gives the flexibility to the system to delete a large set of users if the organization ceases to have a relationship with the data marketplace.
- REQ-UC2-AC5: Data consumers must have permissible purpose to use the data.
- REQ-UC2-AC6: Data exports will be limited to aggregated data and/or model code. Data exported to be evaluated by statistical test to ensure they are anonymized before any data is available for viewing by any user.
- REQ-UC2-W1: The system will be designed to evaluate the data provided by each contributor to determine the relative sensitivity of the data elements provided. These data elements will then be categorized into appropriate buckets to apply appropriate data wrapping techniques.
- REQ-UC2-W2: Based upon the different levels of sensitivity and categorization established in REQ-UC2-W1, the system should enable the selection of various different wrapping techniques that need to be applied to that data (techniques will be determined based upon the level of risk of the data). The goal is to offer flexibility in terms of security and data wrapping approach. This variety in options will provide a secondary level of security from an outward attack.

Comment. Use Case 2 provides structured data and requires the application of protection techniques (wrapping/sanitization) at ingestion. This use case also requires the wrapping of datasets in storage. The protection of data depends on their sensitivity. Data access is regulated according to the intended use of the data. Data owners can select the configuration parameters for sanitization.

2.1.3 Cloud-based data markets for consumer analytics (UC3)

Use Case 3 corresponds to a B2B sensitive data sharing scenario in the cloud. The main goal of this use case is to realize data analytics with sanitization and to achieve formal privacy guarantees for shared data anonymization.

Data

- REQ-UC3-SL2: Storing the result of the anonymization service (i.e., the sanitized data set) on the cloud should be possible for data owners.
- REQ-UC3-SL3: Sharing the result of the anonymization with data consumers (possibly data owners at the same time) through the cloud platform shall be possible.

Policies

- REQ-UC3-AC1: Access control decisions should support selective authorizations given by a data owner at different granularity with respect to the authorized subject.
- REQ-UC3-AC2: It should be possible to group several data owners into a group and grant or revoke access for the entire group.
- REQ-UC3-SL1: Anonymization parameters should be chosen by the data owner. The strength of the anonymization parameters is at the discretion of the data owner. This can be achieved by exposing sanitization services through a fine-grained API to data owners.

Comment. Use Case 3 requires anonymization during ingestion. The sanitized data can be possibly stored in the data market. Data owners can specify anonymization parameters in addition to their data. Access to data is regulated according to authorizations that can be defined at different granularity levels.

2.2 Principles and desiderata

From the requirements listed in the previous section, we derive some principles that the model and language should satisfy, and that guided our decisions.

- *Flexible management of heterogeneous datasets.* The data market should manage data of different kinds, ranging from structured to unstructured data, and accessible at different granularity levels (REQ-UC1-DI4, REQ-UC2-DI2).

  Datasets in the data market can correspond to datasets in their original format, datasets protected through the application of wrapping and/or sanitization techniques, and datasets corresponding to the results of analytics (REQ-UC1-DI5, REQ-UC1-DP1, REQ-UC1-DP2, REQ-UC1-DP3, REQ-UC3-SL2, REQ-UC3-SL3).

- *Fine-grained protection.* Data protection techniques can be applied on datasets at different levels of granularity. The protection techniques can be applied during the different phases of the data life-cycle, ranging from ingestion to analytics (REQ-UC1-DI7, REQ-UC2-W6).

- *Fine-grained access control.* Access control should support different granularity levels with respect to datasets and subjects (REQ-UC1-AC1, REQ-UC1-AC2, REQ-UC2-AC1, REQ-UC3-AC1).

- *Configurable protection.* The owners of the datasets should be able to specify how their datasets should be protected through wrapping or sanitization techniques. The owner can specify the kind of technique as well as the corresponding privacy parameters regulating its working (REQ-UC1-DG3, REQ-UC3-SL1).

- *Purpose-based access control.* Access control should support access and usage restrictions based on purpose (REQ-UC2-AC5).

- *Abstractions.* The policy model and language should support the specification of access restrictions based on typical abstractions (e.g., groups) defined over the domain of users (REQ-UC2-AC2, REQ-UC2-AC3, REQ-UC3-AC2).

  Data owners moving their datasets to the data market should be able to specify whether their datasets can be shared with a particular consumer or set of consumers (REQ-UC1-AC5, REQ-UC1-AC6).

- *Metadata support.* The policy model and language should support the specification of access restrictions based on conditions on metadata describing (meta)properties of the datasets such as the level of sensitivity of (portions of) datasets, and the kind of datasets (REQ-UC1-AC4, REQ-UC2-W2).

In addition to these principles, our work will also take into account further desiderata.

- *Human- and machine-understandable language.* The protection requirements should be specified using a format both human- and machine-understandable that should be simple and expressive. It should be also easily to verify the compliance with respect to defined protection requirements.

- *Extensible model and language.* The model and language should be easy extensible considering new domains, vocabularies, and new requirements.

- *Scalable and efficient implementation.* The model and language should be practically usable in real-word scenarios and applications where the scalability and efficiency in policy specification and enforcement is fundamental.

- *Practical applicability and compatibility with existing technologies.* The model and language should enjoy practical applicability as well as compatibility and integrability with the current approaches and technologies used in the interaction with data market providers, thus ensuring direct deployment.

3. Policy model

The goal of this chapter is to describe the assumptions on the reference scenario as well as of the basic elements of the model and language. Our policy model also considers the metadata model described in deliverable D3.1 "First version of the reference metadata model" [OW20a].

3.1 Reference scenario

The reference scenario includes three main parties: *i) data owners* ingesting their data in the data market; *ii) data market consumers* accessing data in the data market; *iii) data market provider* offering storage and computation services to data owners and consumers. A data owner can be any entity that has authority over data and hence can regulate how they can be accessed by or shared with others. In GDPR terminology, the data owners and market provider of our model correspond to the data controllers and data processor. We use the term owners and provider to maintain generality. Each party can interchangeably act as either a data owner or a consumer at different times, with respect to a specific instance of a data request. Figure 3.1 illustrates the reference scenario enriched with the MOSAICrOWN solutions. The offered services can be based on the presence of *external data processor* that the data market can use for economic reasons. In the following, we use the term *policy subject* to refer to data owners and/or consumers whenever it is not needed to distinguish between these two roles. Policy subjects are entities that use the services offered by the data market and submit or request access to data. For simplicity, in the following we use simply the term subject (instead of policy subject). The data market provides different accesses or *operations* to *datasets* and supports data exchange between owners and consumers, providing the infrastructure for data storage, transfer, and processing. Datasets in the data market can be in their source format or can be the result of a *transformation* aimed at providing a protected/derived version of the dataset.

This deliverable focuses on the problem of empowering data owners to fully enjoy data market services and functionality, enabling them to selectively share data and contribute to data market platforms, while retaining control over their data. In the remainder of the document, we describe the model and language providing owners with such control by enabling them to specify policies regulating data access, use, and processing.

3.2 Basic elements of the model

From our initial analysis, we have identified *subjects*, *transformations*, *datasets*, *metadata*, *operations*, and *purposes* as basic elements of our policy model that should also be captured by the policy language. We now describe these elements in more details.

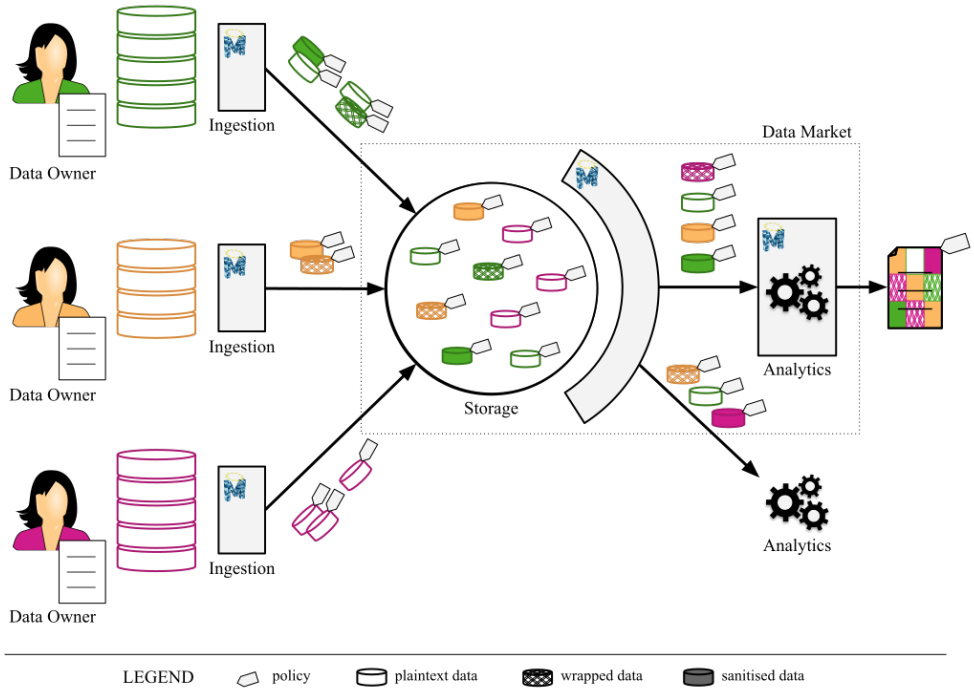


Figure 3.1: Reference scenario

3.2.1 Subjects

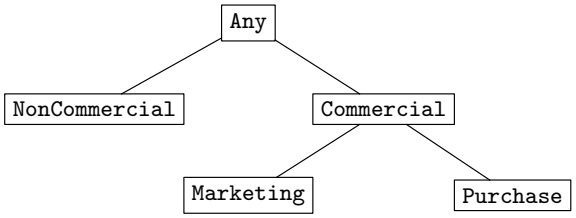
Subjects (data owners and consumers) are entities using the services offered by the data market and submitting or requesting access to data. We use notation S to denote the set of all subjects in the system. Subjects can be independent entities, meaning that they interact with the data market and do not know anything about the other subjects, or can be entities that need to collaborate leveraging the services offered by the data market. The data market recognizes only subjects registered to the market. Each subject is assigned an *identifier* that allows the data market to refer to the subject. Each subject may also have other properties (attributes) associated with it (e.g., name, address). To capture and reason about these properties we assume each subject is associated with a *subject profile* (profile, for short) that characterizes the subject. Figure 3.2(a) illustrates an example of a profile associated with subject `Alice`.

Abstractions can be defined within the domain of subjects that permit to group together subjects with common characteristics and to refer to the whole group with a name. Groups can be nested, meaning that groups can be defined as members of other groups. Subjects together with their groups introduce a *subject hierarchy* H_S that can be represented as a directed acyclic graph whose nodes are the defined abstractions and an arc from node n_i to node n_j indicates a *direct* (i.e., explicitly defined) membership of n_i in n_j. Figure 3.2(b) illustrates an example of subject hierarchy. We assume the subject hierarchy to be rooted, meaning there is one element to which all elements in the hierarchy belong, either directly or indirectly. This assumption is not limiting (a dummy group to which all elements in a domain belong can be assumed) and is common in many systems, where, for example, a group `Public` is usually considered to which all the subjects belong. In the remainder of this deliverable, we assume group `Any` to be the root of the subject hierarchy.

Subject profile

Name	Alice
Address	Merry Street
Citizenship	EU
Email	mymail@myorg.com
Telephone	555-5555
Groups	{`NonCommercial`}
Purposes	Research
Registration Date	2020-05-05

(a)



(b)

Figure 3.2: An example of subject profile (a) and subject hierarchy H_S (b)

Fulfilled principles

- *Fine-grained access control* (REQ-UC1-AC1, REQ-UC1-AC2, REQ-UC2-AC1, REQ-UC3-AC1).
  The possibility of having subjects corresponding to individual users and groups fulfills the need of supporting access rules defined for subjects at different granularity levels.
- *Abstractions* (REQ-UC2-AC2, REQ-UC2-AC3, REQ-UC3-AC2; REQ-UC1-AC5, REQ-UC1-AC6).
  The model explicitly supports the specification of an abstraction over the domain of subjects.

3.2.2 Transformations

Data owners must be able to control and tune the protection applied to their data and to specify how to share them in an easy way. In MOSAICrOWN we distinguish between two main classes of techniques for protecting datasets: *sanitization* and *wrapping*. Both techniques provide some protection to datasets but differ in terms of their application and functionality. Sanitization operates on the source dataset producing a transformed, and less sensitive, version of the source dataset or the result of some computation/analysis on it. Wrapping provides a layer of protection on the data so that data provide for self-protection. The main distinguishing characteristic, stressed in MOSAICrOWN, between the two techniques is that sanitization is *non-reversible* (i.e., it is not possible to go from the sanitized value to the original value), while wrapping is *reversible* (i.e., it is possible, via additional knowledge, function, or key, to retrieve the original value from the wrapped one). To be reversible, wrapping techniques must be based on a *secret* piece of information whose knowledge allows the retrieval of the original data. As an example, techniques based on symmetric encryption can be considered as wrapping techniques since the knowledge of the encryption key allows a subject to retrieve (through decryption) the original data. With respect to the sanitization techniques, they can be further classified depending on the following two main aspects.

- *Global/local:* sanitization techniques can be applied at different granularity levels. There are sanitization techniques that work on a collection of data values, which we call *global* sanitization techniques, and sanitization techniques that work on a single data value, which we call *local* sanitization techniques, where the term local refers to the fact that each value singularly taken is sanitized.

	Local	Global
Deterministic	pseudonymization, generalization, suppression top/bottom coding, noise	mining, noise
Nondeterministic	differential privacy	k-anonymity, ℓ-diversity t-closeness, differential privacy

Figure 3.3: Examples of sanitization techniques

- *Deterministic/nondeterministic:* the application of a sanitization technique many times on the *same data* may always produce the same result or may produce different results. We call *deterministic* the first kind of sanitization techniques and *nondeterministic* the second flavor of sanitization techniques.

Figure 3.3 shows some examples of sanitization techniques classified according to the two aspects above-mentioned. We will use the term *transformation* to refer indistinctly to wrapping and sanitization techniques, and we will use notation T to denote the set of all available transformations in the system.

As we will see in the following, application of sanitization and wrapping can be controlled at the policy level, with the policy language supporting authorizations restricting subjects to access sanitized/wrapped version of the data (in contrast to the original data values). Note that transformations (i.e., wrapping and sanitization techniques) could be applied either: *i) offline*, meaning a sanitized/wrapped view over the data is produced and stored in the data market and such a materialized view is the one available for access, or *ii) on-the-fly* (i.e., the time of access) to sanitize/wrap data to be released to the requester. It is to be noted that global or nondeterministic sanitization techniques enforcing syntactic privacy metrics (e.g., k-anonymity or ℓ-diversity) should, in general, not be applied on-the-fly. The main reason is that such runtime evaluation could result in producing different sanitized views over the same data opening the door to intersection attacks and enabling inferences on the sensitive data behind the sanitization. For instance, the application of k-anonymity, even with the same degree of k, on a same table or on a slightly different version of the table (e.g., where a tuple has been added) could enforce generalization of different attributes, resulting in the release of two views over the table, each of which satisfies the aimed privacy degree but whose combination does not. We also note that the case of differential privacy, which can be global and nondeterministic, can be different. Reason for this is that differential privacy is itself by design a control on the process, rather than on the data produced, and it is aimed to support multiple access on the same data (recomputing the differentially-private result at run-time). Multiple accesses require however to maintain information on the privacy budget (ε) still available for the data [BL20]. With these observations being noted, we do also note that our policy model is agnostic with respect to this and can regulate access on materialized version of sanitized data as well as call for the application of sanitization at run time as dictated by the data owner. Given observations above, considering the integration of sanitization techniques with the policy language and their enforcement at run-time when access is requested, the application of local/deterministic sanitization techniques (operating on the transformation of individual values independently) appears more interesting. In fact, such sanitization can be included in the policy specification and enforced at access time without the risk of improper information leakage.

By analyzing the different kinds of transformations, we can see that each of them is characterized by a *signature* stating the input parameters and the format of the output. More precisely,

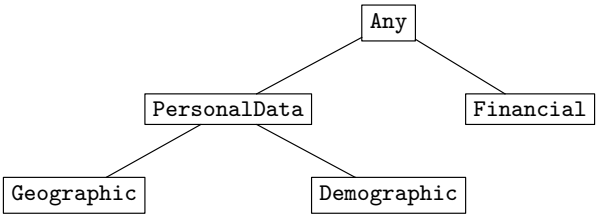


Figure 3.4: An example of data category hierarchy H_D

given a transformation $t \in T$, we model the signature of t as a term of the form **transformation_name**(*arguments*)::*output* where the **transformation_name** identifies the transformation t, *arguments* are the input parameters needed for the application of the transformation, and *output* is the schema of the output produced by the transformation. For example, "**SymEnc**($d.a$, *key*)::$d.a$" denotes the symmetric encryption of attribute a of dataset d with key *key*. Table 3.1 illustrates some examples of transformations on which we will comment later in the chapter.

Fulfilled principles

- *Fine-grained protection* (REQ-UC1-DI7, REQ-UC2-W6).
  Transformations can be applied on datasets during any phase of their life-cycle and can be applied to whole datasets or to single data items.

- *Configurable protection* (REQ-UC1-DG3, REQ-UC3-SL1).
  Transformations can be used by data owners to protect their datasets who can then determine the kind of transformation as well as its protection parameters.

3.2.3 Datasets

Datasets are the resources on which subjects can perform an operation and are stored on the data market platform. We use notation D to denote the set of all datasets in the data market. We consider both *unstructured* and *structured* datasets. An unstructured dataset is characterized by a unique identifier $d \in D$. A structured dataset is characterized by a unique identifier $d \in D$ and by a set of attributes, denoted $d.A$, composing the dataset schema. We use the dot notation to refer to a specific attribute of a dataset, meaning that notation $d.a$ denotes attribute a in the set $d.A$ of attributes of the structured dataset d. We consider accesses to whole datasets (i.e., an access to an unstructured dataset is either allowed or denied) as well as accesses at finer granularity (i.e., sets of attributes of structured datasets).

Datasets can be organized in a hierarchical structure, denoted H_D, defining sets of datasets, called *data category*, that can be referred together with a given name. The definition of categories of datasets introduces a hierarchy of datasets and categories thereof. Figure 3.4 illustrates an example of data category hierarchy. The root of the hierarchy is represented by category `Any`.

Datasets in the data market can be of three different types, as described in the following.

- *Source datasets* are datasets without any protection and correspond to the data generated and/or collected by the data owner. Source datasets are moved to the data market as they are, in their original format. This implies that the data market is trusted for accessing the source datasets and/or that such datasets are not sensitive.

Wrapping techniques

AsymEnc($d.a$, *pkey*)::$d.a$	asymmetric encryption of attribute a of dataset d with public key *pkey*
SymEnc($d.a$, *key*)::$d.a$	symmetric encryption of attributes a of dataset d with key *key*

Sanitization techniques

Gen($d.a$, *level*)::$d.a$	generalize attribute a of dataset d up to *level* levels
AddNoise($d.a$, *noise*, *method*)::$d.a$	add an amount *noise* of noise (in percentage) to attribute a of dataset d using *method*
TopBotCoding($d.a$, *value*, *replacement*, *kind*)::$d.a$	top (kind='top') or bottom ('kind=bot') coding of attribute a of dataset d considering *value* as a limit and *replacement* as the replacement value
Kanon($\{d.\{a_1,\ldots,a_m\}, k$)::$d$	k-anonymity over quasi identifier $a_1,\ldots,a_m$ of dataset d with value k
CountDiff($d.a$, ε, *method*)::$(v_1,c_1)\ldots(v_m,c_m)$	differentially private histogram $(v_1,c_1)\ldots(v_m,c_m)$ over attribute a of dataset d with v_i a value of a and c_i the corresponding count, privacy budget ε and *method* as the method used for adding noise

Table 3.1: Some examples of transformations

- *Transformed datasets* are datasets on which the data owner or the data market has applied a transformation. We talk about: *1) local transformation* when it is performed by the data owner before ingesting the datasets in the data market; *2) market transformation* when it is performed by the market provider; *3) hybrid transformation* when it is performed by the data owner as well as the data market provider.

- *Derived datasets* are datasets obtained as result of analysis or computations performed over other (source and/or transformed) datasets.

Considering the specification of access policies over the datasets in the data market, it is particularly interesting to focus on the transformed datasets. We can distinguish between the *materialized* transformed datasets (materialized datasets, for short) and the *virtual* transformed datasets (virtual datasets, for short). The materialized datasets are typically datasets obtained through the application of wrapping techniques or global nondeterministic sanitization techniques. Materialized datasets are the result of offline sanitization which is explicitly stored in the data market (see previous section). Materialized datasets can then be treated as the source datasets on which we can define access restrictions. Virtual datasets are instead datasets for which there is only a description of how they can be obtained from other datasets but they are not materialized. Such virtual datasets are typically the protected version of source datasets obtained through the application of wrapping techniques or local deterministic sanitization techniques whose corresponding transformation can be computed on-the-fly. In fact, as already discussed in the previous section, the application of such techniques on the same datasets always produces the same result, thus avoiding possible data breaches (which instead may happen with the global nondeterministic sanitization techniques). In the following, given a source dataset $d \in D$, we use notation d^M and d^V to denote a materialized and virtual, respectively, dataset obtained from d. Table 3.1 illustrates some examples of transformations. This table shows the signature of some well-known wrapping and sanitization techniques such as:

- signature **SymEnc**($d.a$, *key*)::$d.a$ corresponding to a wrapping technique that takes an at-

CardHolder.{id, name, surname, dob, g, lr, tag}	*Source dataset* storing information about the holders of credit cards
$\texttt{CardHolder}^M$.{dob, g, lr}	*Materialized dataset* obtained from CardHolder by applying the k-anonymity technique **Kanon**({CardHolder.dob,CardHolder.g}, 5) followed by a projection over attributes dob, g, and lr
$\texttt{CardHolder}^V$.{dob, g, lr}	*Virtual dataset* obtained from CardHolder by applying the symmetric encryption wrapping technique on attribute dob (**SymEnc**(CardHolder.dob), 578hdk%) and attribute g (**SymEnc**(CardHolder.g), 578hdk%) followed by a projection over attributes dob, g, and lr

Figure 3.5: An example of source, materialized, and virtual transformed dataset

tribute $d.a$ and a symmetric encryption key *key* as input and returns the encrypted version of the input attribute;

- signature **Kanon**($\{d.\{a_1,\ldots,a_m\}, k$)::$d$ corresponding to a sanitization technique that takes a set of attributes $(a_1,\ldots,a_m)$ of dataset d, and a privacy degree k as input and returns the same dataset d on which the k-anonymity technique has been applied over the quasi-identifier $a_1,\ldots,a_m$.

Consider now the source dataset CardHolder composed of a set of records of holders of credit cards and characterized by attributes: id (identifier), name (name of the card holder), surname (surname of the card holder), dob (date of birth), g (gender), lr (loan risk), tag (hmac of record). Figure 3.5 illustrates the source dataset and an example of materialized and virtual dataset. The materialized dataset $\texttt{CardHolder}^M$ is the 5-anonymous version of the source dataset CardHolder where the quasi-identifier corresponds to the set of attributes {dob, g}. The virtual dataset $\texttt{CardHolder}^V$ can be obtained from CardHolder by encrypting attributes dob and g.

Fulfilled principles

- *Flexible management of heterogeneous datasets* (REQ-UC1-DI4, REQ-UC2-DI2, REQ-UC1-DI5, REQ-UC1-DP1, REQ-UC1-DP2, REQ-UC1-DP3, REQ-UC3-SL2, REQ-UC3-SL3).
  The model supports any kind of datasets, including datasets resulting from computations. Also, datasets can be stored in the data market in their original format or in a protected way.

- *Fine-grained access control* (REQ-UC1-AC1, REQ-UC1-AC2, REQ-UC2-AC1, REQ-UC3-AC1).
  The model supports the reference to datasets at different granularity levels.

3.2.4 Metadata

Metadata describe the datasets in the data market, and can represent a variety of information including data provenance, data category as well as privacy metrics and degrees (in case of sanitized data), or encryption (in case of wrapped data), as discussed in Deliverable D3.2 [OW20b]. Availability of metadata can be leverage in policy specifications. At high level, the metadata associated

CardHolder

Metadata attribute	Value	Description
creator	companyX	name of the creator
category	Financial	category of the dataset
purpose	Purchase	purpose for which the dataset can be used

(a)

CardHolder^M

Metadata attribute	Value	Description
creator	companyX	name of the creator
category	Financial	category of the dataset
purpose	Purchase	purpose for which the dataset can be used
source	CardHolder	dataset from which the dataset has been obtained
transformation	k-anonymity	transformation used for obtained the dataset
privacy_degree_k	5	privacy degree used in the transformation

(b)

CardHolder^V

Metadata attribute	Value	Description
creator	companyX	name of the creator
category	Financial	category of the dataset
purpose	Purchase	purpose for which the dataset can be used
source	CardHolder	dataset from which the dataset has been obtained
transformation	symmetric encryption	transformation that will be used for computing the dataset on-the-fly
enc_attr	{dog, g}	attributes that will be encrypted on-the-fly

(c)

Figure 3.6: Metadata attributes associated with dataset CardHolder (a), materialized dataset CardHolder^M (b) and virtual dataset CardHolder^V (c)

with a dataset can be modeled as a set of pairs of the form $\langle attribute, value \rangle$. Metadata attributes can be possibly used in the specification of policies.

Metadata associated with a transformed dataset are derived from the metadata of the corresponding source dataset and can be further enriched with other metadata (e.g., with metadata that describe how the transformed dataset has been computed). As an example, consider the datasets in Figure 3.5 and suppose that dataset CardHolder is associated with the metadata reported in Figure 3.6(a). The transformed datasets CardHolder^M and CardHolder^V have their metadata, some of which may be derived from those associated with CardHolder and others can be specific to the transformed datasets, as visible from Figure 3.6(b) and Figure 3.6(c), respectively.

Given a dataset d, we assume the existence of a function META() that makes the association between a dataset and its metadata. For instance, given dataset CardHolder, META(CardHolder) refers to the metadata associated with it. Like for the attributes of datasets, we will use the classical dot notation to refer to a specific metadata attribute. For instance, META(CardHolder).creator denotes the metadata attribute creator associated with dataset CardHolder.

Fulfilled principle

Metadata support (REQ-UC1-AC4, REQ-UC2-W2).

The model supports the management of metadata that describe properties of datasets.

3.2.5 Operations

Subjects can perform different operations, denoted O, on the datasets stored in the data market platform. Operations may vary depending on the specific dataset. Abstractions can also be defined on operations, specializing actions or grouping them in sets. Notation H_O is used to denote the hierarchy built over the set O of operations and rooted at `Any`. There are different kinds of operations that need to be supported such as *download*, *analyze*, and *browse*.

- *Download*, meaning that a subject can download a dataset and can perform off-line analysis. Download can be defined at fine granularity.
- *Analyze*, meaning that a subject can invoke a set of pre-defined operations that perform calculations on selected datasets. The type of analysis can depend on the specific dataset and can be performed according to different modalities: *internal*, the data market provider is trusted to correctly perform the required analysis and to correctly enforce the possible wrapping/sanitization protection of the input datasets and of the result; *external*, a third party outside the data market platform performs the analysis. In this case, the data market platform should possibly enforce the required wrapping/sanitization protection on the datasets on which the external party will perform the analysis.
- *Browse*, meaning that a subject can browse a dataset without downloading it.

In addition to these operations, data owners can perform other operations that allow them to keep control on their datasets as required by possible regulations (e.g., GDPR). These operations include: *delete*, data owner requires to delete her datasets from the data market platform (e.g., a delete operation can be required to enforce the *right to be forgotten* or *right to erasure*); *modify*, data owner requires to modify her datasets (e.g., a modify operation can be required to enforce the *right to rectification*); *access*, data owner requires to access her datasets (e.g., for enforcing the *right of access by the subject*).

3.2.6 Purposes

The processing of datasets should be in line with the expectations of the corresponding data owners regarding how the datasets are expected to be used by the requester. For instance, a data owner may decide to share her datasets only for research purposes. The concept of *purpose* is also a basic principle in recent regulations such as the General Data Protection Regulation (GDPR) where Article 5 (1) (b) mentions the concept of "purpose limitation". Purpose limitation means that the purposes for the processing must be specified. Given a set P of purposes, abstractions can also be defined over them (e.g., see the W3C Data Privacy Vocabularies and Controls CG). Like for the subject hierarchy, different purposes can be grouped together and can be represented by a more general purpose (e.g., `Sale` and `Marketing` purposes can be represented by the `Commercial` purpose). This is equivalent to say that purposes can be organized according to a hierarchy, denoted H_P, which can be depicted as a directed acyclic graph with a root element. Figure 3.7 illustrates an example of purpose hierarchy rooted at `Any`.

Fulfilled principle
Purpose-based access control (REQ-UC2-AC5).
The model supports the concept of purpose.

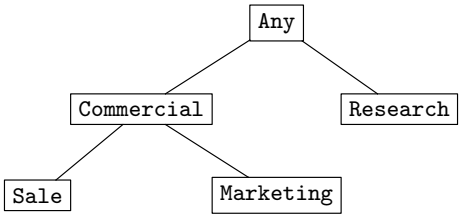


Figure 3.7: An example of purpose hierarchy H_P

3.3 Summary of concepts and notations

We briefly summarize the basic concepts and notations introduced in the chapter and that will be at the basis of the policy language discussed in the next chapter. We report them as a glossary in alphabetical order.

- *Category*. An abstraction of datasets belonging to the data category hierarchy H_D.
- *Customer*. Organization interested in accessing data offered through the data market platform.
- *Data market provider*. Organization that receives data from *data owners* and makes them available to other parties accessing the data market platform.
- *Data owner*. An entity that produces datasets that are made available through the data market platform.
- *Data category hierarchy* (H_D). Abstractions defined over the set D of datasets forming a hierarchy rooted at `Any`.
- *Dataset* (D). Data stored on the data market platform access to which must be controlled.
- *Derived dataset*. Dataset resulting from the analysis of other datasets in the data market.
- *Deterministic sanitization technique*. Sanitization technique such that different applications of such a technique on the *same* data always produces the *same* result.
- *Global sanitization technique*. Sanitization technique that works on a collection of data values.
- *Local sanitization technique*. Sanitization technique that works on a single data value.
- *Materialized dataset*. A transformed dataset obtained through the application of wrapping, global or nondeterministic sanitization techniques that are stored on the data market platform.
- *Metadata*. Information associated with a dataset, describing properties of the dataset or its content.
- *Metadata function* (META()). Function returning the metadata attributes associated with a dataset.

- *Nondeterministic sanitization technique*. Sanitization technique such that different applications of such a technique on the *same* data produces *different* results.
- *Non-reversible*. Property of a protection technique corresponding to the fact that it is never possible to go from a protected value to the original value.
- *Operations* (O). Set of possible operations that data subjects can perform over the datasets in the data market.
- *Operation hierarchy* (H_O). Abstractions defined over the set O of operations forming a hierarchy rooted at `Any`.
- *Purposes* (P). Set of purposes for which datasets can (or will be) used.
- *Purpose hierarchy* (H_P). Abstractions defined over the set P of purposes forming a hierarchy rooted at `Any`.
- *Reversible*. Property of a protection technique corresponding to the fact that it is always possible to go from a protected value to the original value using a secret piece of information.
- *Sanitization technique*. Non-reversible protection technique producing a transformed version of a source dataset.
- *Signature*. It refers to the input parameters and the format of the output produced by a transformation.
- *Source dataset*. Dataset ingested in the data market without any protection stored on the data market platform.
- *Subjects* (S). Set of data owners and consumers of the data market.
- *Subject hierarchy* (H_S). Abstractions defined over the set S of data subjects forming a hierarchy with groups and rooted at group `Any`.
- *Subject profile*. Set of attributes (properties) characterizing a subject registered with the data market platform.
- *Transformations* (T). Set of wrapping and sanitization techniques used for protecting datasets.
- *Transformed dataset*. Dataset obtained from the application of a transformation over source datasets.
- *Virtual dataset*. A transformed dataset that can be obtained through the application of wrapping, local or deterministic sanitization techniques and for which the data market keeps only a description.
- *Wrapping technique*. Reversible protection technique producing a transformed version of a source dataset.

4. Policy language

The goal of this chapter is to provide a description of the first version of the MOSAICrOWN policy language for regulating access to datasets stored in a data market. We first provide a description of the different kinds of policies needed for addressing the identified requirements. We then focus on the access policies and show the different components of the policy rules.

4.1 Types of policies

To address the general principles and desiderata identified in Chapter 2, different types of policies need to be introduced.

- *Ingestion policies.* They specify how data should be protected/stored and determine the basic access policy.
- *Access policies.* They govern access to data by the subjects.
- *Sticky policies.* They govern how the released datasets will be (or should be) managed by the receiving subject.

Ingestion policies. Ingestion policies regulate the policy that will apply to the data when moved to the data market. The ingestion policy can be seen as a meta-policy, since it regulates the specification of access policies. Such policies could check that the generation of access policies does not violate possible restrictions specified over the data. For instance, an ingestion policy could specify that all datasets coming from a specific data owner cannot be stored in plaintext but can only be stored in the encrypted form obtained through the application of a given encryption algorithm.

Access policies. Access policies define policy rules concerning access to datasets in the data market. Policy rules correspond to *positive* authorizations that need to be checked when a subject submits an access request. When a subject request is submitted to the data market platform, the data market first evaluates such request against the authorization policies *applicable* to it. If there exists a policy rule that matches the request, the access is permitted. If none of the policy rules apply to the subject request or there are conditions in the policy rules that cannot be evaluated, the request is denied.

Sticky policies. Sticky policies are policies that *stick* with the data, that is, that follow the data as they move, even when they are exported outside the data market. From a data governance point of view, we are interested in the specification and enforcement of the sticky policy in the data market itself, which needs to ensure that recipients of a data release do receive the sticky policy that must be obeyed on the data. According to their aim and semantics, sticky policies are clearly aimed

at specifying restrictions (in contrast to permissions), restricting future use or further releases and access. The enforcement of the sticky policy itself outside the data market environment is not under our scope.

Note that the first version of the language deals with only access policies: ingestion policies and sticky policies will be the subject of future work.

4.2 Access policy components

We are now ready to present the first version of the access policy language that fulfills the requirements discussed in Chapter 2. We first describe the format of a subject request and then illustrate the different components of the policy rules.

4.2.1 Subject request

The data market provider receives requests from subjects for processing datasets (we use the term "processing" in a broader sense, to indicate any operation that is performed on datasets including storage, reading, analytics, and so on). A subject request is a quadruple of the form:

$$\langle \textit{subject}, \textit{dataset}, \textit{operation}, \textit{purpose} \rangle$$

- *subject* is the subject that makes the request.
- *dataset* is the dataset on which *subject* wants to perform *operation*.
- *operation* is the operations that is being requested.
- *purpose* is an optional component of the request and represents the reason for which dataset *dataset* is being requested.

Subject. A subject can be either an entity registered with the data market platform or can be an anonymous entity. In the first case, *subject* is an identifier from which it is possible to retrieve the corresponding subject profile registered with the data market. In the second case, the subject corresponds to the reserved keyword `anonymous`.

Dataset. The *dataset* component of a request can be of two different forms: *1)* identifier d of a dataset, meaning that the request requires the access to the whole dataset; *2) view* over a dataset d defined as a subset of the attributes of dataset d, that is, $\{d.a_1, \ldots, d.a_m\}$ with a_i an attribute of dataset d (i.e., $a_i \in d.A$). Note that, for the sake of simplicity, the view over a dataset *id* can also be expressed as $d.\{a_1, \ldots, a_m\}$.

Operation. The *operation* component of a request corresponds to the operation that the *subject* requires to perform over *dataset*. Clearly, actions to be considered may vary depending on the specific dataset. Broadly speaking, we can distinguish between two main classes of operations:

- operations for accessing or managing datasets as they are. Examples of this class of operations are `download`, `read`, and `delete`.

- operations for updating or analyzing datasets which may produce a derived dataset that will be stored on the data market.

For this version of the language, we mainly focus on the first class of operations and therefore we assume that the *operation* component of a subject request corresponds to the name/identifier of an operation.

Purpose. The purpose component corresponds to an element of the purpose hierarchy H_P (Section 3.2.6). One of the motivations for having the purpose in the request arises from the GDPR saying that "the data subject has given consent to the processing of his or her personal data for one or more specific purposes". Considering the purpose hierarchy in Figure 3.7, examples of correct values for the component *purpose* are: `Research` and `Commercial`.

The following are examples of subject requests.

- ⟨`anonymous`, `read`, `Branch`, `Any`⟩: an anonymous subject requires to `read` dataset `Branch` for `Any` purposes. The dataset includes public information about the branches of a financial institution.
- ⟨`Alice`, `download`, `CardHolder`, `Commercial`⟩: subject `Alice` requires to `download` dataset `CardHolder` for `Commercial` purposes.
- ⟨`Alice`, `access`, `CardHolder`.{`name`, `surname`}, `Marketing`⟩: subject `Alice` requires to `access` (with `access` a generalization of both `download` and `read` operations) attributes `name` and `surname` of dataset `CardHolder`.

4.2.2 Access policy rule

A policy is composed of a set of *policy rules* that regulate how the datasets stored in the data market can be used by requesting subjects. A policy rule has the following form.

$$\langle subject, dataset, operation, purpose \rangle$$

This rule says that all subjects *matching* the *subject* can perform operation *operation* over dataset *dataset* for purpose *purpose*. We now analyze in more details the different components of the policy rule.

Subject. This component of the rule allows the reference to a set of subjects possibly depending on conditions defined over the attributes of a subject's profile. More precisely, a *subject* is a pair of the form: ⟨*id*, *condition*⟩, where *id* is the identifier of a subject or the name of an abstraction in H_S, and *condition* is a Boolean formula of simple conditions "*attribute_name op value*" with *attribute_name* the name of an attribute of a subject profile, *op* a standard built-in mathematics operator, and *value* a value compatible with the domain of the attribute. Note that the *condition* component is optional. Intuitively, a *subject* is evaluated with respect to the subject making a request. To make it possible to refer, in the *subject*, to the subject of the request being evaluated without the need of introducing variables in the language, thus allowing an efficient evaluation of conditions, we introduce the keyword **subject**. Appearance of keyword **subject** in a *condition* component is intended to be substituted with the actual parameters of the request in the evaluation at access time. For instance, **subject**.`citizenship` indicates the attribute `citizenship`

within the profile of the subject whose request is being processed. The *subject* then evaluates to true if both the *id* component and the *condition* component over the profile of the requester evaluate to true. Note that whenever the *condition* contains a simple condition that cannot be evaluated with respect to the profile of the requester, we consider the overall evaluation as "undefined" and the *subject* evaluates to false. The following are examples of subjects.

- ⟨`Any`,_⟩ denotes any subject.
- ⟨`NonCommercial`, **subject**.`citizenship`=EU⟩ denotes subjects who are member of the group `NonCommercial` and are European citizens.
- ⟨`Any`, **subject**.`country`=Italy⟩ denotes any subject who lives in Italy.

Dataset. This component of the rule allows the reference to a set of datasets possibly depending on whether they satisfy given conditions that are evaluated on the metadata associated with the datasets. More precisely, a *dataset* may refer to datasets at different granularity levels: it may refer to a dataset as a whole as well as to any of its components. For this latter, as discussed in Chapter 3, we assume structured data characterized by a set of attributes (our model can be easily extended to the consideration of semi-structured or unstructured data). More precisely, a *dataset* can be a pair of the form ⟨*id*, *condition*⟩ where *id* is the identifier of a dataset, the name of a data category in H_D (Section 3.2.3), or a view of the form $d.\{a_1,\ldots,a_m\}$ with $\{a_1,\ldots,a_m\}\subseteq d.A$, and *condition* is a Boolean formula of simple conditions "*meta_attribute_name op value*" with *meta_attribute_name* the name of a metadata attribute associated with a dataset, *op* a standard built-in mathematics operator, and *value* a value compatible with the domain of the metadata attribute. Like for the *subject* component, the *condition* element is optional. Note that to make it possible to refer to the metadata associated with the dataset to which the request being processed refers, without the need of introducing variables in the language, we introduce the keyword **metadata**. The appearance of such a keyword in *condition* is intended to correspond to the set of metadata attributes associated with the dataset specified in the request.

The datasets referenced in the *dataset* can be any of the three types of datasets mentioned in Section 3.2.3, meaning that the policy rule may also refer to transformed (materialized or virtual) datasets, thus defining a rule for a protected version of a dataset. As an example, consider the dataset `CardHolder` and the corresponding transformed datasets in Table 3.5. The following are example of *dataset*.

- ⟨`PersonalData`, **metadata**.`creator` = companyX⟩ denotes datasets of category `PersonalData` that have been created by companyX;
- ⟨`Financial`, (**metadata**.`method` = k-anonymity ∧ **metadata**.`anonymization_degree_k` ≥ 5) ⟩ denotes datasets of category `Financial` that have been anonymized by applying k-anonymity with $k \geq 5$;
- ⟨`Any`,_⟩ denotes all datasets stored in the data market platform that belong to a category defined in the category hierarchy H_D;
- ⟨$\texttt{CardHolder}^V$.{`dog`, `g`}, _⟩ denotes attributes `dob` and `g` of the virtual transformed dataset $\texttt{CardHolder}^V$ where these attributes are encrypted.

Keywords	`anonymous`	
Reserved identifiers	**subject**	bounded to the identity (if defined) of the subject making a request
	dataset	bounded to the identifier of the dataset to which access is requested
	metadata	bounded to the identifier of the metadata associated with the dataset to which access is requested

Table 4.1: List of keywords and reserved identifiers of the language

Note that in addition to the reserved identifier **metadata**, we also introduce the identifier **dataset** that indicates the identifier of the dataset to which access is requested.

Table 4.1 summarizes the list of keywords and reserved identifiers of the language.

Operation. This component of the rule corresponds to the operation to which the policy rule refers. Note that abstractions can also be defined on operations, specializing operations or grouping them in sets. An *operation* corresponds then to the identifier of a basic operation, or the identifier of an abstraction. For instance, an *operation* can be `read`, `download`, `Any` (root of the operation hierarchy denoting any operation), or `access` assuming that `access` groups both `read` and `download`.

Purpose. This component of the rule allows the reference to a specific purpose or to an abstraction defined in the purpose hierarchy H_P (Section 3.2.6). Considering the purpose hierarchy in Figure 3.7, examples of purposes are: `Any` (denoting any purpose), `Commercial`, `Research`, and `Sale`.

4.2.3 Matching a subject request to policy rules

Given a subject request $\langle sbj, ds, op, p \rangle$, we now describe the process matching the request to the policy rules to determine whether the request can or cannot be granted. We assume to use a *close policy* as default, meaning that a subject request is permitted only if there exists at least one policy rule *applicable* to the request. Let $\langle sr, dr, or, pr \rangle$ be a policy rule. Intuitively, a policy rule applies to a subject request when the subject (*sbj*), dataset (*ds*), operation (*op*), and purpose (*p*) of the subject request match the corresponding components in the policy rule. We now describe how the matching between the different components of the subject request and the corresponding components in a policy rule is performed.

Subject matching. The subject *sbj* of the subject request can be *1)* the `anonymous` keyword or *2)* the identifier of a subject registered with the data market platform. The matching works as follows.

1. *sbj*=`anonymous`. The subject *sr* in the policy rule matches the subject in the request if and only if *sr*=$\langle$`Any`,_$\rangle$, with `Any` the root of the subject hierarchy, meaning that the policy rule has been specified for any subject registered with the data market platform.

2. *sbj* is the identifier of a subject registered with the data market platform. The subject component *sr* in a policy rule corresponds to pair $\langle id,condition \rangle$, where *condition* is not mandatory. The matching happens if *id*=*sbj* or *id* is the identifier of a group and *sbj* is a (direct or indirect) member of *id*, and the subject profile associated with *sbj* satisfies *condition* (if any).

Example 1
Subject request
⟨Alice, CardHolder.{name, surname, lr}, read, Commercial⟩
Policy rules
⟨Alice, CardHolder.{name,surname}, read, Commercial⟩
⟨Alice, CardHolder.{dob, g, lr}, read, Commercial⟩

Example 2
Subject request
⟨Alice, CardHolder.{dob, g, lr}, read, Commercial⟩
Policy rules
⟨Alice, CardHolder^{M}.{dob, g, lr}, read, Commercial⟩
⟨Alice, CardHolder^{V}.{dob, g, lr}, read, Commercial⟩

Figure 4.1: Examples of dataset matching

Dataset matching. The dataset *ds* component in a subject request can be: *1)* the identifier of a dataset; or *2)* a view defined over a specific dataset (i.e., a subset of the attributes characterizing the dataset). The matching works as follows.

1. *ds* is the identifier of a dataset stored in the data market platform. The dataset *dr* in the policy rule matches the dataset in the request if and only if *dr*=⟨*id*, *condition*⟩ with *i) id*=*ds* or *id* the identifier of a data category such that *ds* (directly or indirectly) belongs to such category, *ii)* the metadata attributes associated with *ds* satisfy the *condition* (if any).

2. *ds*=*d*.$\{a_1,\ldots,a_m\}$ with $\{a_1,\ldots,a_m\}$ a subset of the attributes of dataset *d*. The dataset *dr* of a policy rule matches a view-based request over attributes $\{a_1,\ldots,a_m\}$ of dataset *d* if and only if *dr*=⟨*id*, *condition*⟩ with *id*=*d* or *id* the identifier of a data category such that *d* (directly or indirectly) belongs to such category, or *id*=*d*.$\{a_{i_1},\ldots,a_{i_n}\}$, with $\{a_1,\ldots,a_m\} \subseteq \{a_{i_1},\ldots,a_{i_n}\}$, and metadata attributes META(*d*) associated with *d* satisfy *condition*. In other words, in case of a view-based request, a policy rule matches the request if and only if the policy rule has been defined for the same dataset, for a data category to which the requested dataset (directly or indirectly) belongs, or for a superset of the attributes of the same dataset appearing in the request. This definition implies therefore that the view-based access over a dataset must be permitted by an individual policy rule, thus avoiding side-effects that may be caused by an improper combination of policy rules [DFJ+08]. Chapter 5 will illustrate how such a view-based matching has been enforced through the joint attribute visibility property.

To better understand the rationale behind the view-based matching, consider the examples in Figure 4.1. With respect to Example 1, the subject request submitted by Alice states that Alice wishes to read attributes name, surname, and lr of dataset CardHolder for Commercial purposes. By submitting such a request, Alice is also implicitly asking to read the association of a given person (name and surname) with the corresponding loan risk (lr). However, none of the two policy rules in Example 1 allows Alice to know such an association. In fact, the first policy rule allows Alice to know the name and surname of all card holders while the second policy rule allows Alice to know the association among dob, g, and lr. With respect to Example 2, the subject request submitted by Alice states that Alice wishes to read attributes dog, g, and

`lr` of dataset `CardHolder` for `Commercial` purposes. Although one may argue that the two policy rules allow `Alice` to read a protected version of attributes `dog`, `g` together with attribute `lr` (i.e., a generalized version of these attributes, with the first policy rule, and an encrypted version of these attributes, with the second policy rule, as described in Table 3.5) and that the request could be permitted and translated into a request over the protected version of the attributes, the request is instead denied. The reason is that the two policy rules are in conflict because they allow the access to different views over attributes `dog` and `g` and the selection of one view over the other one cannot be automatically done by the data market platform. To access the protected version of these attributes, `Alice` should then submit a request specifying $\texttt{CardHolder}^M$ or $\texttt{CardHolder}^V$ as the dataset involved in the request.

Operation matching. The operation *op* of the subject request matches the operation *or* of the policy rule if and only if *op*=*or* or *op* is a (direct or indirect) descendant of *or* in the operation hierarchy H_O.

Purpose matching. The purpose *p* of the subject request matches the purpose *pr* of the policy rule if and only if *p*=*pr* or *p* is a (direct or indirect) descendant of *pr* in the purpose hierarchy H_P.

4.3 Ongoing work and other aspects

The research results obtained in the first period of MOSAICrOWN are important steps towards the main goal of designing an efficient and expressible policy model and language. Work on the policy model and language is continuing along different directions, including *restrictions* and *policy derivations*, as discussed below. Restrictions are policy rules aimed at restrict data access (in contrast to permitting). Restrictions can be useful to limit the scope of authorizations (an can be particularly appealing for sticky policy specifications). Policy derivation would allow the derivation of specifications from the ones explicitly given, for instance to automatically enable subjects authorized to access a dataset to also access a sanitize version of it.

In addition to the work on the policy model and language, Work Package 3 has investigated other aspects related to the problem of empowering data owners with control in the digital data market. We briefly illustrate them below.

We have proposed a new approach for supporting data owners in the specification of requirements and preferences on services [DFL+19]. The proposed approach is simple to use also by non-specialists and makes it possible to define in a flexible way different protection requirements that may need to be imposed on datasets. The basic idea consists in allowing data owners to specify their protection requirements using high-level concepts/parameters and to automatically compare them against the low-level characteristics of services. The mapping between high-level concepts and parameters and low-level characteristics is performed by a brokering service through a set of implication rules, dictating which values for parameters imply which values of high-level concepts. Intuitively, an implication rule identifies the combinations of values assumed by configuration parameters that imply a given value for a concept. In particular, the broker evaluates the available services against the desiderata of the data owner to assess the level of satisfaction in the different services. The brokering process operates in two phases, the first one abstracting to concepts, and the second one performing assessment.

Another line of research has been looking at the management of confidentiality constraints in Description Logics ontologies (*DL-Lite*$_{\mathcal{R}}$ ontologies). Many open interesting challenges charac-

terize this scenario. Also, alternative semantics for the protection have been investigated, producing tractability results for them [LRS19, CLRS20]. These results can lead to the application of these expressive protection models to large scale systems.

5. Policy engine

Together with the policy model and language, we have investigated possible approaches to enforcement. Here, we present some investigation towards the *policy engine* tool that is responsible for parsing the MOSAICrOWN policies and checking whether a data subject request is permitted. We first describe the encoding of our policy language in ODRL (Section 5.1). Then, we illustrate the *joint attribute visibility* property considered in the evaluation of access requests to take access control decisions (Section 5.2). We describe the technologies at the basis of the implementation of the policy engine and how it performs query analysis (Section 5.3). Finally, we describe the ongoing work (Section 5.4).

5.1 Policy language

MOSAICrOWN policies are expressed in ODRL [IV18], a W3C policy specification language that provides a flexible and interoperable information model, vocabulary, and encoding mechanisms for representing statements about the usage of content and services.

The choice of ODRL has been motivated by a number of considerations. ODRL has been originally designed to manage the requirements of platforms for the distribution of digital content (e.g., songs, movies). This scenario is a clear example of a wide market for digital data, where market operators let producers and consumers meet. The management of restrictions on the use of the content is of great importance for this scenario and ODRL supports an expressive representation of these restrictions. ODRL also is based on a model that is well designed and flexible. Compared to access and usage control models implemented by specific platforms and operating systems, the model is designed to be extensible, supporting its adaptation to scenarios that go beyond the representation of access privileges to multimedia files. The specification of ODRL already provides an attempt at the definition of a Privacy Profile, with an explicit consideration of the adaptability of ODRL to the MOSAICrOWN scenario. ODRL is also designed to be a high-level specification, with a variety of representations. Similar choice has been made for the MOSAICrOWN policy language, which considers alternative representations, with the one based on JSON-LD described in this chapter as the main option, but not the only one. For instance, an OWL representation is already being developed, to facilitate the application of reasoning on the policy itself.

The use of ODRL has given the opportunity to immediately adapt to MOSAICrOWN a number of components that have already been developed for ODRL, like the examples of specific vocabularies for the representation of operations and subjects. A feature of ODRL is that it is not directly related to a specific engine for the execution of the policy. This is another aspect where there is alignment with the MOSAICrOWN policy language, which aims to operate in a variety of technological domains, with data collections represented in many formats, like JSON-LD objects, tables in classical relational databases, Data Frames supported by the Apache Spark big data frameworks. The efficiency requirements deriving from the management of large data collections mandate the design of a number of ad-hoc solutions for each domain.

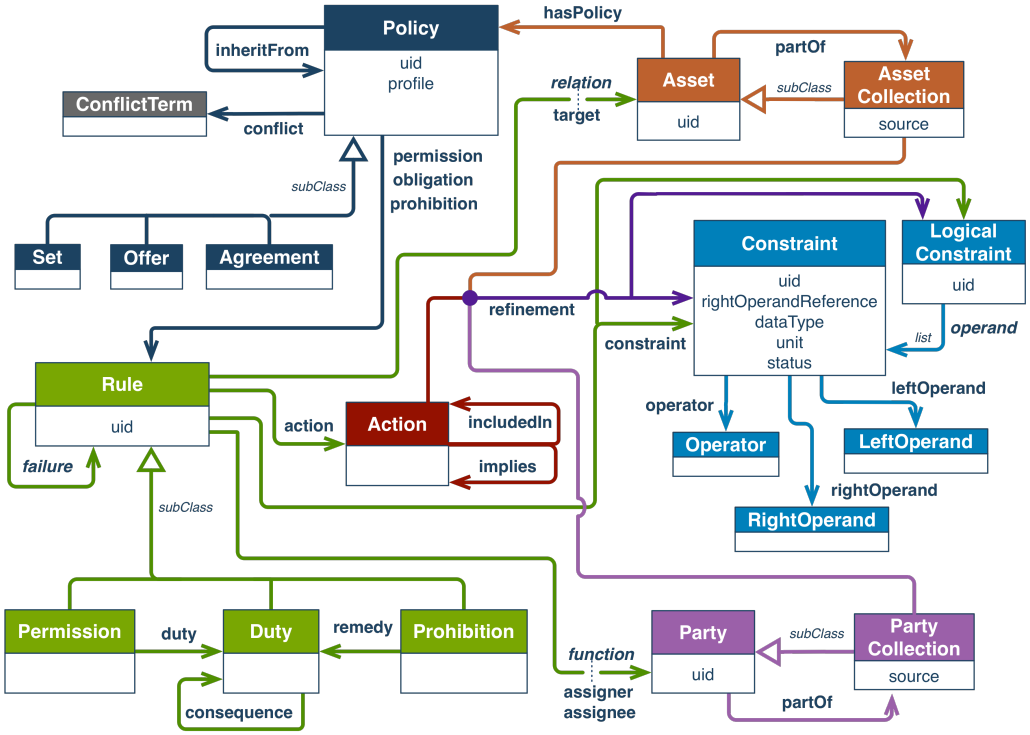


Figure 5.1: ODRL information model [IV18]

The major limitation of ODRL is the relatively limited support it receives in current digital multimedia markets. We argue this is due to the fact that the specification has been recently completed and the ambitious design requires a dedicated effort from platform designers to be applied. These markets also tend to be isolated, reducing the benefit that derives from the use of a W3C standard. The interoperability is instead a major requirement for the MOSAICrOWN scenario and the use of an existing specification is a significant benefit for the proposal.

We now describe how the MOSAICrOWN policy language can be mapped onto ODRL. In the previous chapter, we have shown that a policy rule consists of four components:

$$\langle subject, purpose, dataset, operation \rangle$$

Considering the broad and flexible ODRL specification model in Figure 5.1 [IV18], the components of the MOSAICrOWN policy rules can be mapped onto the following ODRL entities:

- *subject* corresponds to **assignee**;
- *dataset* corresponds to **target**;
- *operation* corresponds to **action**;
- *purpose* corresponds to **purpose**.

Each of the four entities composing a policy rule in ODRL is uniquely addressed by a *Uniform Resource Identifier* (URI) [Ber], a short string that permits to identify any resource on the web. In this way, any kind of information about each of the entities can be retrieved through subsequent HTTP requests. This also permits the policies to be compliant with the *Linked Data*

```
{
  "@context": [
    "http://www.w3.org/ns/odrl.jsonld",
    "https://www.mosaicrown.eu/ns/mosaicrown.jsonld"
  ],
  "uid": "http://example.com/policy/1",
  "permission": [
    {
      "uid": "http://example.com/policy/1/1",
      "assignee": "http://example.com/user/Alice",
      "target": [
        "http://example.com/table/CardHolder/name",
        "http://example.com/table/CardHolder/surname"
      ],
      "action":  "read",
      "purpose": "statistical"
    }
  ]
}
```

Figure 5.2: An example of an ODRL policy expressed in JSON-LD

Principles [W3Cc], as they can provide introspective information by the use of vocabularies or simply by referencing external sources.

ODRL specification can be implemented using different serializations: *XML* (*eXtensible Markup Language*), *RDF* (*Resource Description Framework*), *OWL* (*Web Ontology Language*), and *JSON* (*JavaScript Object Notation*). Among these, JSON permits to encode linked data using *JSON-LD* (*JSON for Linked Data*) [W3Ca]. JSON-LD is a way to create a network of standard-based, machine-readable data across web sites. Starting from a single piece of linked data, it is possible to retrieve other linked data that are hosted on different sites across the Web simply following the links to them. This key feature, in addition to the ease of reading and writing, makes JSON-LD the most suitable way to express MOSAICrOWN policies. Figure 5.2 illustrates an example of ODRL policy expressed in JSON-LD. The policy states that `Alice` can `read` attributes `name` and `surname` of `CardHolder` for `statistical` purposes.

Note that the use of URIs for identifying the targets in policy rules permits to include any kind of target (i.e., dataset) in the rules. For concreteness and simplicity of description of our policy engine, in this chapter we will focus on structured data and specifically consider relational databases in our examples. The target of policy rules will then be relations and/or attributes.

Since an official ODRL engine (or processor) [W3Cb] is missing, we have implemented a policy engine processing the MOSAICrOWN policy dialect. Given the importance of this activity from a security point of view, and the need of the engine to access all the policy rules, our preliminary implementation is based on the assumption that the policy engine operates in a trusted environment, which is in line with the scenarios considered in MOSAICrOWN. In the following of this chapter, we discuss the first version of the policy engine tool. Specifically, we show how the parsing process is implemented, how joint attribute visibility and rule constraints are addressed, and how policy rules are verified. We also describe the techniques adopted to enforce access control, focusing in particular on query evaluation.

CardHolder

CustomerID	Name	Surname	DateOfBirth	Email	Marketing	CreditScore
342940	John	Smith	2000-01-31	jsmi@example.org	TRUE	650
304198	Jane	Doe	1995-06-08	jdoe@example.org	FALSE	800
257834	Mario	Bianchi	1989-10-08	mbia@example.org	FALSE	720
...	...	...	...	...	...	...

Transaction

TransactionID	CustomerID	Day	Month	Year	Amount	Merchant
68748367	342940	18	04	2020	34.50	Medical Center Ltd.
62748368	304198	03	01	2020	104.00	Shopping Mall S.A.
65748369	257834	07	03	2020	12.43	Groceries GmbH
...	...	...	...	...	...	...

Figure 5.3: An example of two relational tables with a color-coded representation of the target of the policy rules in Figure 5.4

5.2 Joint attribute visibility

A core concept of our model is the *joint attribute visibility* property, which states that if two or more attributes of the same relational table can be accessed together, they must appear together in a policy rule. In other words, the association among different attributes of a same relational table is visible if and only if all attributes appear together in the same policy rule. For instance, if `Alice` can read from a relational table attribute a_1, attribute a_2, or their association (a_1,a_2), we have to define a single policy rule including both attributes in the target component. The definition of two policy rules, one for attribute a_1 and one for attribute a_2, does not have the same effect since `Alice` would not be authorized to access the two attributes together. Note that joint attribute visibility property applies only to attributes belonging to the same relational table. The access to attributes of different relational tables can instead be regulated by different policy rules. The basic idea is that different policy rules may allow the joint visibility of attributes of different relational tables whenever such policy rules grant access to the attributes that can be used to join the tables, allowing a subject to perform a join operation between the two tables. By imposing this condition for the join operations, we also limit the information that can be obtained by submitting two requests on different relational tables and then performing the join operation at the client side.

As an example, consider the two relational tables in Figure 5.3 and the policy rules in Figure 5.4 defined over the attributes of the two relational tables. These tree policy rules allow `Alice` to `read`, for `statistical` purposes, attributes:

- `CustomerID` and `DateOfBirth` of table `CardHolder` (first rule);
- `CustomerID`, `Month`, and `Year` of table `Transaction` (second rule);

```
{
  "@context": [
    "https://www.w3.org/ns/odrl.jsonld",
    "https://www.mosaicrown.eu/ns/mosaicrown.jsonld"
  ],
  "uid": "http://example.com/policy/1",
  "permission": [
    {
      "uid": "http://example.com/policy/1/1",
      "assignee": "http://example.com/user/Alice",
      "target": [
        "http://example.com/table/CardHolder/CustomerID",
        "http://example.com/table/CardHolder/DateOfBirth"
      ],
      "action": "read",
      "purpose": "statistical"
    },
    {
      "uid": "http://example.com/policy/1/2",
      "assignee": "http://example.com/user/Alice",
      "target": [
        "http://example.com/table/Transaction/CustomerID",
        "http://example.com/table/Transaction/Month",
        "http://example.com/table/Transaction/Year"
      ],
      "action": "read",
      "purpose": "statistical"
    },
    {
    "uid": "http://example.com/policy/1/3",
    "assignee": "http://example.com/user/Alice",
    "target": [
      "http://example.com/table/Transaction/Month",
      "http://example.com/table/Transaction/Year",
      "http://example.com/table/Transaction/Amount",
      "http://example.com/table/Transaction/Merchant"
    ],
    "action": "read",
    "purpose": "statistical"
    }
  ]
}
```

Figure 5.4: An example of a policy with three rules

- `Month`, `Year`, `Amount`, and `Merchant` of table `Transaction` (third rule).

Assume now that `Alice` submits a request for reading attributes `CustomerID` and `Merchant` of table `Transaction` for statistical purposes. This request would be denied because no policy rule allows the joint visibility of these two attributes (`CustomerID` belongs to the target of the second rule and `Merchant` to the target of the third rule). On the other hand, a request over attributes `CustomerID`, `Month`, and `Year` of `Transaction` would be permitted (second rule); the same holds for a request over attributes `Month`, `Year`, and `Merchant` of `Transaction`

(third rule). The policy rules defined for `Alice` allow her to perform some analysis on the customers (e.g., count of transactions per cardholder per month), and on the merchant (e.g., sum of transaction amount grouped by merchant and month) separately. Furthermore, according to the joint visibility property, these policy rules authorize `Alice` to see the association between attributes `CustomerID` and `DateOfBirth` of table `CardHolder` and attributes `CustomerID`, `Month`, and `Year` of table `Transaction`. In fact, the first and second policy rules have attribute `CustomerID` in common (which is a *foreign key* for table `Transaction`) that can be used to perform a join between the two tables. `Alice` cannot instead access attribute `DateOfBirth` of `CardHolder` together with `Merchant` of `Transaction` as the latter cannot be seen together with the join attribute.

5.3 Policy engine implementation

The *policy engine* is written in Python and performs several tasks associated with parsing and importing MOSAICrOWN policies and deriving policy knowledge graph based on them (i.e., an entity-relationship graph representation of the policy rules). After the policies have been imported, access requests (i.e., queries) can be analyzed against the knowledge graph to verify if they are compliant with the policies. To perform these tasks, the policy engine makes use of the *RDFLib* [Tea20] library. *RDFLib* permits to parse policies written in *RDF* [CWL+14] and *JSON-LD* [W3Ca]. The ODRL vocabulary [ISMR18] is available in these formats and can be directly imported in the RDFLib knowledge graph. Moreover, it is possible to import several namespaces and vocabularies in the knowledge graph that RDFLib generates. We can then import both the ODRL namespace and vocabulary (on which MOSAICrOWN policies are based) as well as the specific MOSAICrOWN namespace and vocabulary to introduce the semantic that is not already available in ODRL, such as the purpose attribute assigned to policy rules.

To verify whether a request is compliant with the policies, we need a mechanism for extracting from the policy knowledge graph all policy rules. To this purpose, we use *SPARQL* [HS13], a semantic query language able to retrieve and manipulate data stored in an RDF graph and integrated in RDFLib. Figure 5.5 illustrates an example of SPARQL query that extracts all the MOSAICrOWN policy rules from the parsed policy.

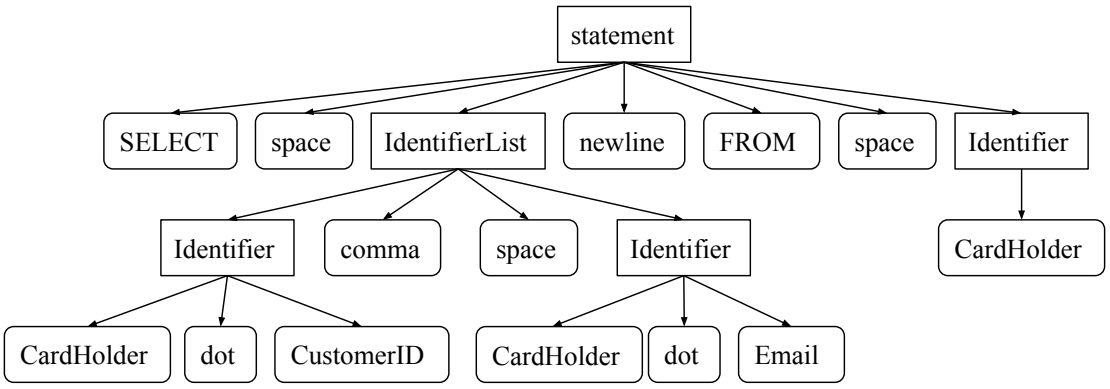
```
PREFIX odrl: <https://www.w3.org/ns/odrl/2/>
PREFIX mosaicrown: <https://www.mosaicrown.eu/ns/mosaicrown/1/>
SELECT DISTINCT ?assignee ?action ?target ?purpose
    WHERE {
        ?policy odrl:permission ?rule .
        ?rule odrl:assignee ?assignee .
        ?rule odrl:action ?action .
        ?rule odrl:target ?target .
        ?rule mosaicrown:purpose ?purpose
    }
```

Figure 5.5: An example of *SPARQL* query that extracts all the rules from a policy

To verify whether a request is permitted, the policy engine is initialized with a set of policies that create a knowledge graph as discussed above. When a query, together with an assignee, an action, and a purpose is formulated, it is checked by the policy engine that, according to the given

```
SELECT CardHolder.CustomerID, CardHolder.Email
FROM CardHolder
```

(a)



(b)

Figure 5.6: An example of SQL query (a) and corresponding Parse Tree obtained through the execution of *sqlparse* (b)

policies, returns one of the following access decisions:

- the query is *permitted*;
- the query is *denied*.

In this first version of the policy engine we focuses on SQL queries. However, the pipeline has been written in such a way that it is possible to extend the set of supported languages by integrating different front-end modules.

We now describe in more details the two steps composing the policy engine pipeline: *i)* query parsing; and *ii)* policy rule verification.

5.3.1 Query parsing

The first step of the policy engine pipeline parses the query and identifies the data necessary to compute the result.

The front-end module parses the SQL queries with *sqlparse* [Alb], a non-validating SQL parser for Python. It provides support to parse the SQL statements, creating their *Parse Tree* (*PT*) representation, which is a concrete representation of the input that preserves all the information, formatting included. For instance, the application of *sqlparse* to the SQL query in Figure 5.6(a) produces the Parse Tree in Figure 5.6(b).

As it is visible from the example of query, we use fully qualified names for attributes (e.g., `CardHolder.CustomerID`) to avoid the collisions of the names. This allows our front-end module to extract the necessary information from the query without the need to have previous knowledge of the relation schema and guarantees uniqueness in the representation of the query targets.

Once the front-end module has obtained the PT representation of the query, it traverses the tree to discover the query targets. The tree traversal keeps track of the relational tables involved in the query (exploring the comma separated list in the `FROM` clause) and identifies the attributes in the:

- comma separated list in the `SELECT` clause;
- expressions following the `ON` keyword and in the `WHERE`, and `HAVING` clauses;
- comma separated lists in the `GROUP BY` and `ORDER BY` clauses;
- comma separated lists within brackets following the name of the function (function parameters);
- structured statements that extensively use expressions (`CASE` clause).

This information is represented in an Abstract Syntax Tree (AST), which is a tree representing the abstract syntactic structure of the query. The root node represents the query itself, while its child nodes represent its sub-queries. The tool supports sub-queries in the `SELECT`, `ON`, `WHERE`, and `HAVING` clauses, in the list of tables expressions, and in the function parameter. The AST supports the identification of the scope of tables and aliases, facilitating the understanding of the overall query structure.

Although the front-end module understands the query structure, when the module provides the targets to the subsequent steps of the pipeline, we have decided to treat the query as a whole, without splitting the accessed attributes by sub-query. While too restrictive, this strategy prevents circumventions of the join attribute visibility constraint that can happen by abusing the query syntax to make attributes appear in different sub-queries. Our solution therefore does not verify how the query is written but only operates on the accessed data. This implies that, even if a query is actually composed of two (or more) queries that do not join their data, our approach will be over-restrictive, as the attributes are considered all together. However, since such a query could be split into two queries by the requester, we are not limiting the utility.

To integrate the front-end module with the policy engine (Section 5.1), targets need to be represented as Uniform Resource Identifiers (URIs). To this purpose, there is the need of providing a mapping from table names to URIs, so that attribute names can be appended to the *URI* segment. Figure 5.7 shows a mapping of table names to URIs and the corresponding URI representation of the targets for the example in Figure 5.6(a).

As already discussed, Figure 5.7 shows that the adoption of fully qualified names permits to map different attributes with the same name that belong to the schema of different tables (e.g., `CustomerID`) to exactly one URI, removing any ambiguity on the target of the query.

5.3.2 Policy rule verification

We now describe how policy rules are verified. For each attribute appearing in the request, the engine extracts from the policy knowledge graph all the policy rules that grant visibility to the assignee, for the specific purpose and action of the request, leveraging the *SPARQL* query shown in Figure 5.5. Policy rules are then grouped by the table to which each targeted column belongs. For each table, the engine checks whether the joint attribute visibility (Section 5.2) is satisfied by at least one policy rule. If such control does not succeed, the access request is *denied*. Otherwise, the access request is *permitted*.

Table	URI
`CardHolder`	http://example.com/table/CardHolder
`Transaction`	http://example.com/table/Transaction

(a)

Table	URI
`CardHolder`	http://example.com/table/CardHolder

Attribute	URI
`CustomerID`	http://example.com/table/CardHolder/CustomerID
`Email`	http://example.com/table/CardHolder/Email

(b)

Figure 5.7: An example of mapping of table names to URIs (a) and of URI representation of the targets of the query in Figure 5.6(a) (b)

5.4 Concluding remarks

In this chapter, we have presented the first version of the MOSAICrOWN policy engine. Our tool supports the parsing of rules in MOSAICrOWN policies and supports access control decisions for the queries issued by subjects. The work on the policy engine is proceeding to include all features of the data model, in particular conditions and transformations (wrapping and sanitization), described in the previous chapter. Also, enrichment of the model and language planned for the next phase of the project, such as restrictions and derivations, will also be included in the policy engine.

6. Conclusions

This document has presented the first version of the policy specification language and model for MOSAICrOWN. The policy work is at the center of the data governance framework to which Work Package 3 is devoted, as it enables data owners to specify (and then have enforced) policies regulating access, use, processing, and release of their data. The model and language have been designed with the goal to ensure simplicity and easiness of use on one side, and expressiveness and flexibility on the other side, as highlighted in Chapter 1. The document has presented the policy work, discussing also the rational behind the choices taken, from requirements to implementation.

Chapter 2 has analyzed the requirements for the three use cases of the project, identifying their impact on the policy model and language. The requirements have been categorized and commented to determine the principles and desiderata to be considered in the work on policies.

Chapter 3 has introduced the policy model, defining the basic concepts and elements. The model responds to the principles and desiderata identified and nicely accounts for inclusion in the policy specification of wrapping and sanitization techniques developed in Work Packages 4 and 5, respectively. While simple in its design, the model enjoys expressiveness, with support for abstractions on the different concepts, conditions on metadata and subjects' profiles, as well as explicit consideration of purpose of use (responding not only to requirements from the use cases, but also to data privacy regulations).

Chapter 4 has provided a preliminary version of the policy language, which assumes policy specification to remain at a simple high-level, to provide for simplicity and easiness of use and understanding for final users.

Chapter 5 has presented our investigation towards the realization of the policy engine enforcing the policy specifications according to our model. First step of such investigation addressed the translation of the high-level policy specifications in terms of interoperable standard rules, considering ODRL. The goal was to provide a policy engine that provides efficient policy evaluation and enjoys interoperability with current technology so to enable its deployment in real-life scenarios.

Bibliography

[ACK+10] C.A. Ardagna, J. Camenisch, M. Kohlweiss, R. Leenes, G. Neven, B. Priem, P. Samarati, D. Sommer, and M. Verdicchio. Exploiting cryptography for privacy-enhanced access control: A result of the PRIME project. *Journal of Computer Security (JCS)*, 18(1):123–160, January 2010.

[ADF+12] C.A. Ardagna, S. De Capitani di Vimercati, S. Foresti, S. Paraboschi, and P. Samarati. Minimising disclosure of client information in credential-based interactions. *International Journal of Information Privacy, Security and Integrity (IJIPSI)*, 1(2/3):205–233, 2012.

[Alb] A. Albrecht. python-sqlparse. https://sqlparse.readthedocs.io/.

[BDF+18] E. Bacis, S. De Capitani di Vimercati, S. Foresti, S. Paraboschi, M. Rosa, and P. Samarati. Protecting resources and regulating access in cloud-based object storage. In I. Ray, I. Ray, and P. Samarati, editors, *From Database to Cyber Security: Essays Dedicated to Sushil Jajodia on the Occasion of his 70th Birthday*. Springer, 2018.

[Ber] T. Berners-Lee. The need for a universal syntax. https://www.w3.org/Addressing/URL/uri-spec.html.

[BK19] P.A. Bonatti and S. Kirrane. Big data and analytics in the age of the GDPR. In *Proc. of the 2019 IEEE International Congress on Big Data*, Los Angeles, CA, USA, December 2019.

[BL20] D. Bernau and G. Livraga, editors. *D5.2 - First report on privacy metrics and data sanitisation*. Technical report of MOSAICrOWN, June 2020.

[BS02] P.A. Bonatti and P. Samarati. A uniform framework for regulating service access and information release on the web. *Journal of Computer Security (JCS)*, 10(3):241–271, 2002.

[BS03] P. Bonatti and P. Samarati. Logics for authorizations and security. In J. Chomicki, R. van der Meyden, and G. Saake, editors, *Logics for Emerging Applications of Databases*. Springer-Verlag, 2003.

[CLRS20] G. Cima, D. Lembo, R. Rosati, and D.F. Savo. Controlled query evaluation in description logics through instance indistinguishability. In *Proc. of the 29th International Joint Conference on Artificial Intelligence (IJCAI 2020)*, Yokohama, Japan, July 2020.

[CWL+14] R. Cyganiak, D. Wood, M. Lanthaler, G. Klyne, J.J. Carroll, and B. McBride. RDF 1.1 concepts and abstract syntax. Technical report, W3C recommendation, 2014.

[DFJ+08] S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati. Assessing query privileges via safe and efficient permission composition. In *Proc. of the 15th ACM Conference Conference on Computer and Communications Security (CCS 2008)*, Alexandria, Virginia, USA, October 2008.

[DFJ+10] S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati. Encryption policies for regulating access to outsourced data. *ACM Transactions on Database Systems (TODS)*, 35(2):12:1–12:46, April 2010.

[DFL+19] S. De Capitani di Vimercati, S. Foresti, G. Livraga, V. Piuri, and P. Samarati. A fuzzy-based brokering service for cloud plan selection. *IEEE Systems Journal (ISJ)*, 13(4):4101–4109, December 2019.

[DFLS20] S. De Capitani di Vimercati, S. Foresti, G. Livraga, and P. Samarati. Towards owners' control in digital data markets. *IEEE Systems Journal (ISJ)*, 2020. to appear.

[DFS20] S. De Capitani di Vimercati, S. Foresti, and P. Samarati. Specification and enforcement of access policies in emerging scenarios. In J. Vacca, editor, *Cloud Computing Security: Foundations and Challenges, 2nd Edition*. CRC Press, 2020. to appear.

[HS13] S. Harris and A. Seaborne. SPARQL 1.1 query language, 2013. https://www.w3.org/TR/sparql11-query.

[ISMR18] R. Iannella, M. Steidl, S. Myles, and V. Rodriguez-Doncel. ODRL vocabulary & expression 2.2, 2018.

[IV18] R. Iannella and S. Villata. ODRL information model 2.2, 2018. https://www.w3.org/TR/odrl-model.

[JSSS01] S. Jajodia, P. Samarati, M.L. Sapino, and V.S. Subrahmanian. Flexible support for multiple access control policies. *ACM Transactions on Database Systems (TODS)*, 26(2):214–260, June 2001.

[LRS19] D. Lembo, R. Rosati, and D.F. Savo. Revisiting controlled query evaluation in description logics. In *Proc. of the 28th International Joint Conference on Artificial Intelligence (IJCAI 2019)*, Macao, China, August 2019.

[OW20a] A. O Mahony and R. Wenning, editors. *D3.1 - First version of the reference metadata model*. Technical report of MOSAICrOWN, March 2020.

[OW20b] A. O Mahony and R. Wenning, editors. *D3.2 - Preliminary version of tools for the governance framework*. Technical report of MOSAICrOWN, May 2020.

[Tea20] The RDFLib Team. RDFLib: A Python library for working with RDF, 2020. https://github.com/RDFLib/rdflib.

[W3Ca] W3C JSON-LD Working Group. JSON-LD, JSON for Linking Data. https://json-ld.org/.

[W3Cb] W3C Permissions and Obligations Expression Working Group. Requirements for an ODRL Processor. https://www.w3.org/2017/05/18-poe-minutes.

[W3Cc] W3C Wiki. LinkedData definition. https://www.w3.org/wiki/LinkedData.

[WB19] R. Wenning and D. Bernau, editors. *D2.1 - Requirements from the Use Cases*. Technical report of MOSAICrOWN, December 2019.

[ZDX+20] Y. Zhang, R. Deng, S. Xu, J. Sun, Q. Li, and D. Zheng. Attribute-based encryption for cloud computing access control: A survey. *ACM Computing Surveys*, 2020. (to appear).